

mistralXG – a USB connected, PIC-based MIDI synthesizer

Part I

Introduction

mistralXG is a MIDI synthesizer, based on a MIDI daughter card such as the Yamaha DB50XG. It could also be used as a USB MIDI switching and monitoring device without the daughtercard.

In this first of two articles, I'll explain how mistralXG operates and discuss the main technologies it uses. The second article will look at the hardware and software in more detail. Parts cost for the project is \$20 to \$30, excluding the daughter card, power supply and enclosure.

Choosing a project

A couple of my earlier projects were based on the PIC12C508A, one of the least powerful Microchip PIC microcontrollers (MCUs). Developing them was great fun and provided my first experience of these fantastic little devices. For my next project, I wanted to use a more powerful MCU and utilize some of the wide range of on-chip peripherals.

I've dabbled in MIDI for many years and, as my current PC's sound card hasn't got the necessary WaveBlaster-compatible connector, had an unused Yamaha DB50XG synthesizer daughter card. The Internet revealed some designs that use the DB50XG as a stand-alone synthesizer, but that just feed MIDI data to the card. I wanted something a little more capable, to allow me to play my MIDI wind controller away from my computer. This was an ideal area for my project.

In August 2005, N&V had published Robert Lang's Midi-nator design, which showed that a hobbyist could create a USB-based MIDI device. But, rather than use Midi-nator as a starting point, I wanted to create a complete new MIDI implementation so that I knew exactly what the device was and wasn't capable of.

Start at the beginning

Late in 2007, I began thinking about the design. I burned the MIDI-nator project code into a PIC18F2550 to see it in action and, while it worked well, it was evident that there was a steep learning curve to designing with USB. Thankfully, there is plenty of information available (see references). Microchip also provides comprehensive datasheets – the PIC18F2550 document runs to 430 pages – so I had a lot of reading to do. A few months later, after sketching out some different approaches, I started serious work on the design.

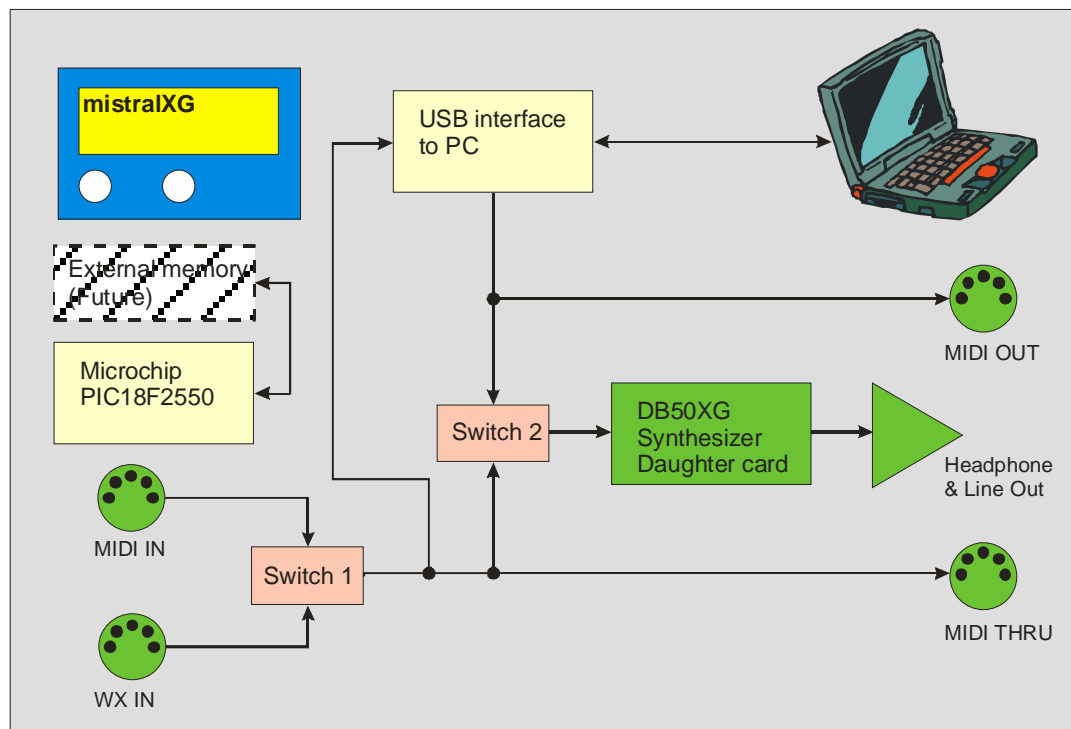


Figure 1. *mistralXG* block diagram

Referring to Figure 1, the PIC18F2550 is the heart of *mistralXG*. It manages data flowing between the MIDI ports and the PC via its serial and USB ports, and provides logic and control for the MIDI switches and the user interface (two push-buttons and an LCD display).

The code offers these features:

- Selecting which MIDI stream is sent to the synthesizer.
- Running Status control for MIDI OUT.
- MIDI data filtering.
- MIDI OUT and MIDI THRU control.
- Adjustment of LCD brightness.
- Monitoring of MIDI IN and MIDI OUT activity.
- Maintaining and displaying error information.

Technology overview

Let's take a brief look at some of the technologies used in the design. If you really want to know how *mistralXG* works, there's no substitute for diving in and reading up on this stuff (pointers to additional information are given in the Resources sidebar). There are also copious comments in the source code, which will be available on the Nuts & Volts website.

MIDI, the Musical Instrument Digital Interface

MIDI is a serial protocol for passing messages between controllers and music synthesizers. Commands such as "Note on", "Bend pitch", "Note off", and "Change instrument" control the sounds a synthesizer makes. Sixteen channels (1-16), each mapped to a single instrument, are defined for a single MIDI connection, but MIDI does not define which instrument each channel represents. MIDI composers have to decide the instrument to channel mapping for each tune.

A command might say, "Play Middle C on channel 5, medium loud". Three bytes are required to construct this command. These are (in hexadecimal) 0x94, 0x3C and 0x40. The first four bits of 0x94 (the "9") indicate this command is "Note on". The "4" in 0x94 says that the note should be played on channel 5 (counting starts at 0 for channel 1). Byte 2 selects note number 60 (0x3C), which is middle C. Notes 0-127 are defined (from low to high pitch). The last byte gives the volume, with 0 being silent and 127 (0x7F) being loudest.

Instruments are numbered 1-128 (0-127) and mappings are again arbitrary. The General MIDI specification, however, defines a mapping most synthesizers can be set to match. When playing a MIDI file, you have to make sure that your synth uses the correct mapping to prevent notes meant for one instrument being played by another – perhaps even the drum kit! One synthesizer's piano may sound different to another's, but at least they both generate the right type of sound for the music.

MIDI hardware recognizes command bytes because their most significant bit (MSb) is set to 1. Command bytes are followed by 0, 1 or 2 data bytes, which have their MSb=0. Details of the commands and how to interpret them can be found in the resource links.

MIDI is real-time

MIDI data are transmitted in real time. As soon as a synthesizer receives a completed "Note on" message for a particular note and channel, it starts to play the note. It will continue to sound until a "Stop note" command for the same note and channel is received. So a synthesizer connected to a MIDI keyboard will reproduce the exact timing of the notes as they are played by the musician, with a slight, usually indiscernible, delay to allow for constructing, transmitting and receiving the MIDI messages. If multiple MIDI devices are chained together, this delay can become significant and must be kept in mind when setting up complex MIDI systems.

MIDI was launched about 25 years ago, and the specification has changed little since then. The data or bit rate of 31.25 kHz seems very slow by today's standards, but MIDI's wide adoption within the electronic music industry has ensured that its survival, and it is likely to be around for many years to come.

USB, the Universal Serial Bus

PCs used to have a variety of "legacy" ports (serial, parallel, etc.), carried over from the first PC shipped in the early 1980's. Setting these up to attach peripherals such as printers was often a configuration nightmare. Now, USB has made connecting even very sophisticated devices to your PC almost trivial.

The end-user's gain has been at a cost to the developer. Designing serial or parallel port hardware is straightforward, but getting a peripheral to communicate using USB is not for the faint-hearted. It requires a complex combination of hardware and software. Fortunately, Microchip has made it easier by providing a few microcontrollers with USB hardware. They also provide a software framework that does most of the housekeeping needed to make USB work. This allows the designer to focus on just the application specific USB code. In practice, however, the designer needs a good understanding of USB to be able to debug problems as they arise during development.

Tell me what you are

When you first connect a USB device to a computer, it goes through a process called enumeration. This tells the computer what the device is and what drivers are required, among

other things. When enumeration completes successfully, the device becomes available to the system. Suitable USB MIDI drivers are provided with both Windows and Linux.

USB data are collected into packets and transmitted in 1ms “frames” (high-speed USB 2.0 uses shorter “microframes”). Thus there is a mismatch with the real time needs of MIDI. As a result, some MIDI purists won’t use a USB MIDI implementation but, in practice, this does not usually cause a problem, especially for simple MIDI configurations.

Liquid Crystal Display (LCD)

Several LCD display types are available. Many, including the one used here, are based on the Hitachi HD44780 controller. This chip supports a number of different display configurations, from eight characters in one row, up to four rows of 20 characters. *mistralXG* uses a 16x2 display.

Four or eight data lines (number selected in software) and three command lines give you complete control of the display. Using only four lines (4-bit mode) for data makes it an attractive proposition for MCU-based designs, where pin counts can be at a premium.

A few simple routines provide all the function needed to write messages to the display. They take care of all the critical timing requirements so that the rest of the code doesn’t have to worry about them.

Microchip PIC18F2550 microcontroller

Microchip offers an impressive range of MCUs that incorporate a wide variety of on-chip peripherals. These include serial ports, A-D converters, timers, EEPROM memory, and more. One less common peripheral is the 2550’s USB port. It is USB 2.0 compatible, supporting both the full-speed (12 Mbps) and low-speed (1.5Mbps) modes but not the high-speed (480 Mbps) mode. MIDI’s 31.25kbps data rate means that the full-speed mode is sufficiently fast for *mistralXG*.

Software is needed to drive the USB hardware so that your PC can enumerate and recognize a valid USB device. Fortunately, Microchip’s USB Firmware Framework does a lot of the work for you, leaving you to concentrate on your application code.

The *mistralXG* code is around 5-6 KB, much smaller than the 16 KB available in the 2550. I chose this device, rather than its smaller sibling, the PIC18F2455, as I wasn’t sure how large the code would grow. It also means there’s plenty of room to add additional features at a later date, such as the external storage shown in Figure 1.

Putting it all together

Ok, so that’s a whistle-stop tour of the main technologies used in *mistralXG*. Now let’s look at how the user interacts with the unit, leaving the technical detail to the next article. As a taster, though, you can see the prototype in Figure 2.

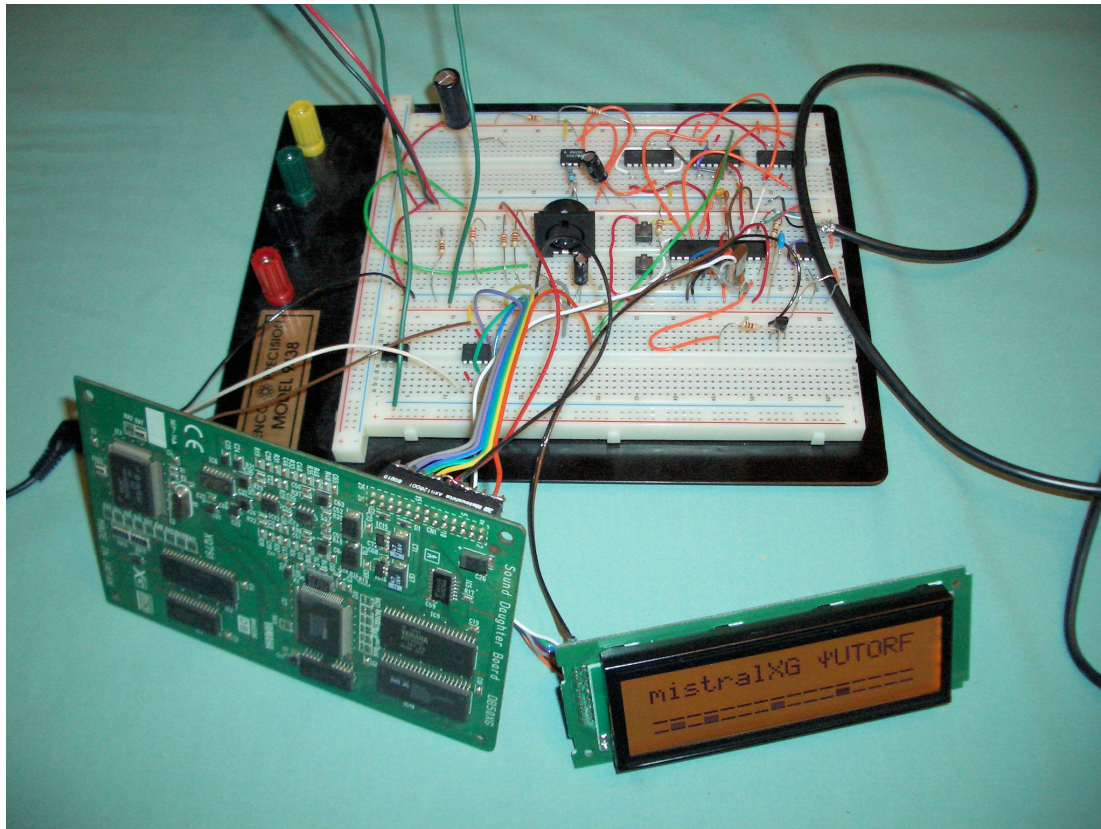


Figure 2. mistralXG prototype

At front left, you can see the DB50XG, with the LCD to the right. The PIC18F2550 is the large chip to center-right of the breadboard.

The user's view of mistralXG is four MIDI ports (MIDI IN, WX IN, MIDI OUT, and MIDI THRU), a USB port, the LCD display, and two push buttons (*Select* and *Set*). When power is applied, a Splash Screen shows the firmware revision and copyright notice (you may use the code for personal projects, but not commercially). After a few seconds, the display changes to the Home Screen, shown in Figure 3.



Figure 3. mistralXG Home Screen

The Home Screen displays:

- USB status: the “cactus”, six characters from the end of the first row, is my lo-res attempt at the USB symbol. It is displayed when mistralXG is connected to your PC.
- The next five symbols show the status of the user options, discussed later.
- Error status: the space to the left of the USB symbol shows the global error flag, and is normally blank. If an error occurs, this changes to an “!” and the error screens become available.
- Real-time MIDI activity monitor: the second row shows MIDI IN (top line) and MIDI OUT (bottom line) activity. In Figure 2, shows data being transmitted from the PC on channels 1, 3, 5, 6, 7 and 10 (left to right). No data is being transmitted to the PC (all channels are low on the top line). The MIDI IN monitor remains active even when mistralXG is not connected to your PC.

The remaining screens allow you to access user options and are displayed by pressing the **Select** button. The **Set** button changes settings or performs some other useful action. From the Home and Splash screens, **Set** provides quick access to useful features (see Figure 4). No matter which screen is being displayed, after a few seconds without a button being pressed, the display returns to the Home Screen.

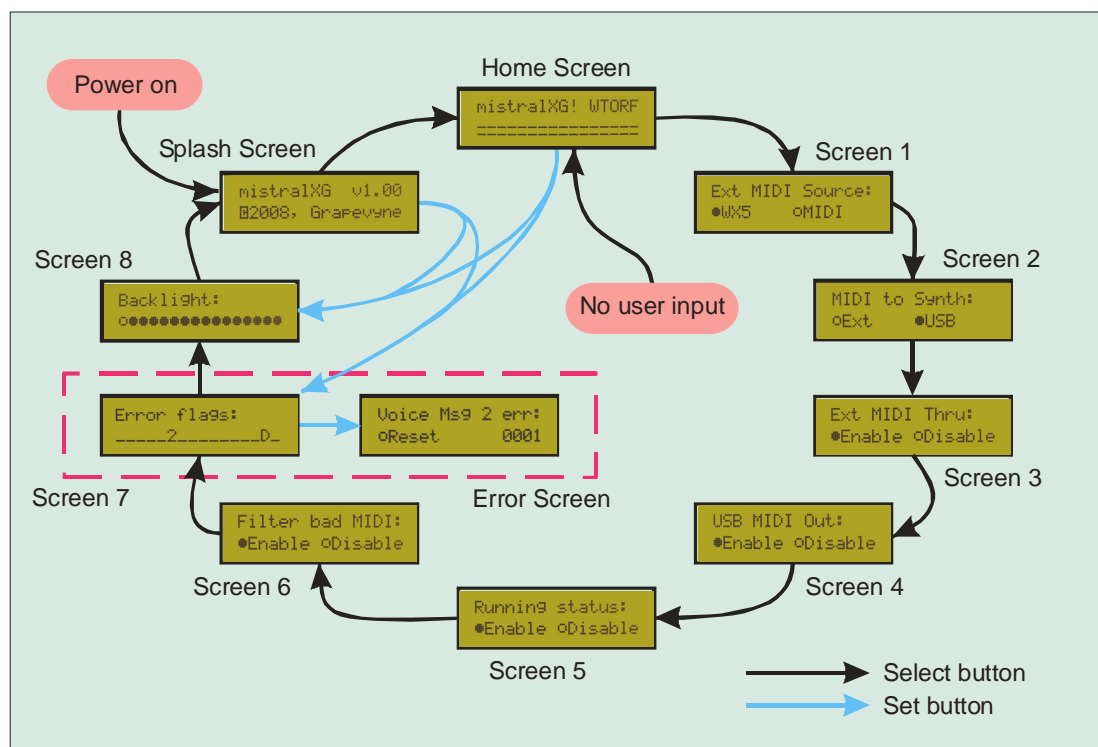


Figure 4. Accessing user options screens

Using mistralXG

Select cycles through the various screens, as shown in Figure 4. Pressing **Set** toggles or otherwise modifies the setting for each particular screen. I’ve already discussed the Home Screen, so now let’s take a look at the others.

User options

The first two screens allow you to select which MIDI stream is transmitted to the synthesizer. The first option flag on the Home Screen (“U” in Figure 3) shows which stream has been selected. Screen 1 lets you choose between the MIDI IN and WX IN inputs (Switch 1 in Figure 1), while Screen 2 selects between the output of Switch 1 and USB data from the PC (Switch 2 in Figure 1). The display shows “M”, “W”, or “U” depending on your choices. If USB is selected and mistralXG is not plugged into your computer, Switch 2 automatically selects your “M”/“W” choice, reverting to “U” when the USB connection is reactivated.

Screens 3 and 4 control the MIDI THRU and MIDI OUT streams respectively. When enabled, MIDI THRU receives the “M” or “W” stream coming from Switch 1. MIDI OUT echoes the MIDI stream coming from the PC. Selecting Disable in these screens switches their respective streams off.

Screen 5 controls MIDI Running Status. This feature of MIDI reduces the amount of data transmitted. It works on the principle that if a status byte would be repeated it can be omitted. So, if consecutive MIDI commands are all “Note on, channel 5” (0x94 as already seen), then the command byte is omitted for subsequent notes until a new status byte is required. This reduces the amount of data transmitted by 10 to 15%. Normally, Running Status should be enabled, but you could disable it if you think it may be causing problems for a receiving device.

Certain byte sequences are invalid MIDI commands. A receiving device should ignore these, so filtering shouldn’t be necessary. If you think invalid messages may be causing problems, however, the filter can be enabled in Screen 6. Invalid sequences are reported as errors by mistralXG, whether or not the filter is active.

Screen 7 is not normally displayed. It shows the error types that have occurred, and only becomes available when an error has been registered. In this case, the Home Screen global error flag is displayed (see Figure 4). Pressing *Set* while on Screen 7 shows the individual error screens. A second press of *Set* resets the specific counter.

After zeroing a counter, pressing *Set* once more allows you to clear all error counters and return to the Home Screen. If you’d rather examine the individual error counts, *Select* cycles through those error screens that have non-zero counters. When all counters have been reset, the global error flag is cleared. Individual error screens are exited by allowing the display timeout to return you to the Home Screen or by choosing to reset all counters.

The LCD backlight brightness is controlled in Screen 8. Nine levels, from Off to Maximum, are available.

Finally, we return to the Splash Screen and then the Home Screen. Figure 4 also shows short cuts to the backlight and error screens using the *Set* key from the Home Screen.

All option settings, including backlight level, are saved in non-volatile memory and reloaded when mistralXG is switched on.

Error flags and counters

mistralXG tracks a number of different errors that may occur. The second line of Screen 8 (the error flags) shows the status of the 16 error types. An underscore (“_”) in a position indicates that the related error has not occurred. Each position has its own error character. If every error type had occurred, the flags would look like this:

BFOD12012SE459DM

Here's a quick summary of the error types. More detailed information is in the source code:

BFO	These flags indicate errors receiving MIDI data from the inputs (receive buffer overflow, EUSART framing error, EUSART overflow error).
D	Unexpected data byte (MIDI command byte expected)
12	Unexpected channel command byte (commands expecting one and two data bytes respectively).
012	Unexpected System Control command byte (commands expecting zero, one and two data bytes respectively).
S	Unexpected Sysex command byte
E	Unexpected End of Sysex command byte
459D	Invalid command byte (0xF4, 0xF5, 0xF9, 0xFD)
M	USB-MIDI input buffer overflow

Errors are rare, but you can induce them by switching the input stream between “M” and “W” while a track is playing.

Next time: Technical details

OK, that about wraps things up for this month. I hope you can see that mistralXG offers a lot of flexibility, and that you are interested enough to want to try either building it for yourself or using it as a base for your own project.

Next time, I'll take you through the details of the hardware and software that bring mistralXG to life.

(SIDEBARS ON NEXT PAGE)

Resources:

Here are a few links that I have found useful. Links come and go of course – if that happens, just load a few key words into your favorite search engine and you'll find lots out there.

MIDI

MIDI specifications: <http://www.midi.org> (unfortunately, these have to be purchased, but web sources are almost as good).

There are hundreds of MIDI-related sites on the Internet. These are just a couple of the top hits from Google:

<http://ccrma-www.stanford.edu/~craig/articles/linuxmidi/misc/essenmidi.html>

<http://www.computermusicresource.com/MIDI.Commands.html>

USB

USB specifications: <http://www.usb.org>. Here you can find both the USB 2.0 spec (650-odd pages!) and more specific documents, including the USB-MIDI spec.

USB in a Nutshell: <http://www.beyondlogic.org/usbnutshell/usb1.htm>

LCD

There are many web sites describing these popular displays. Here are just a couple:

<http://www.doc.ic.ac.uk/~ih/doc/lcd/>

<http://mil.ufl.edu/imdl/handouts/lcd-faq.htm>

PIC18F2550

PIC MCU information: <http://www.microchip.com>

PIC discussion group: <http://www.piclist.com>